PARALLEL PERFORMANCE IMPROVEMENT IN MRGENESIS AND RATPENAT

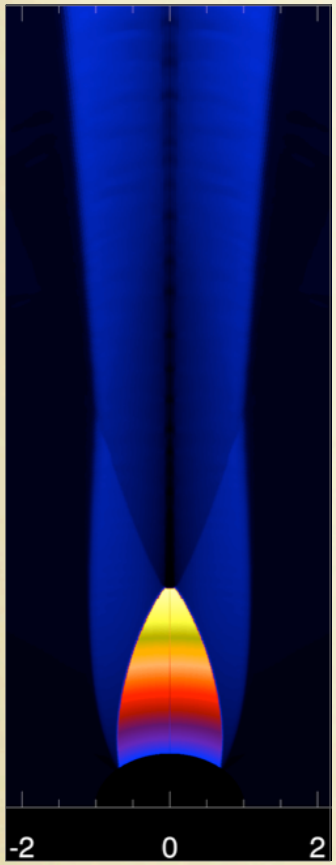
Carmen Aloy Torás

Astronomy and Astrophysics. University of Valencia.

OUTLOOK

- **Efficiency** in relativistic hydrodynamics codes is a key part of the implementation as it allows obtaining results faster.
- The **parallelization** of these codes improves efficiency.
- RATPENAT and MRGENESIS codes were developed by members of the Relativistic Astrophysics Group. The parallelization of these codes is implemented to improve efficiency in terms of **execution times**.

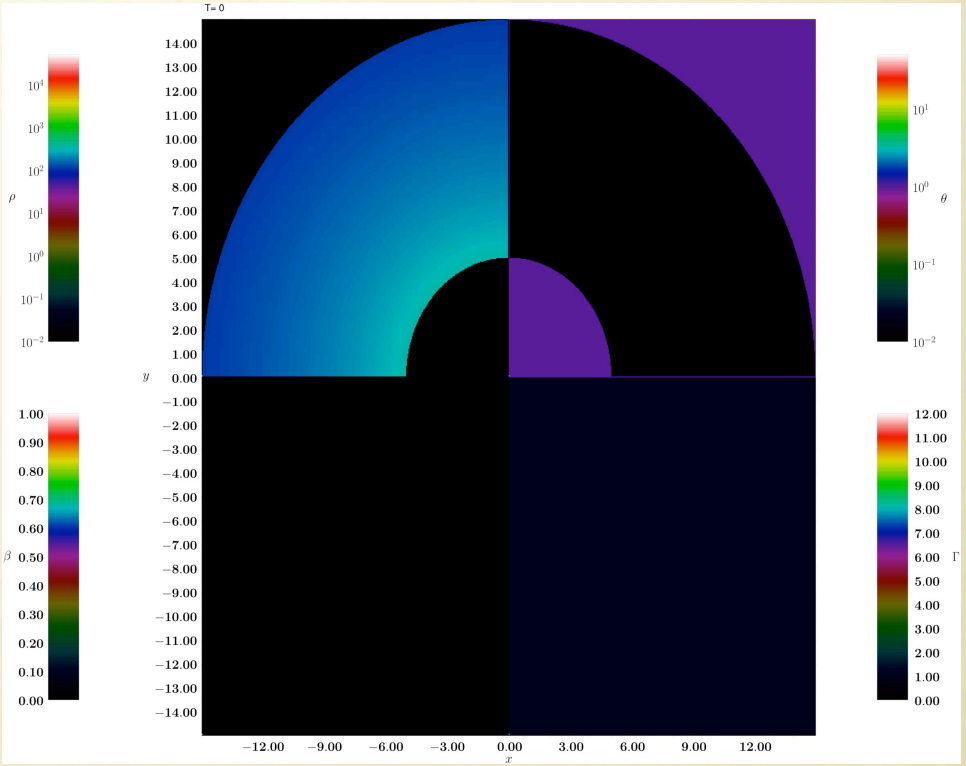
MRGENESIS



The code MRGENESIS employs a finite volume approach in order to evolve the Relativistic Euler equations in presence of dynamically relevant magnetic fields.
This is combined with a Constrained Transport scheme to account for the divergence free evolution of the magnetic field.

Incorporates:

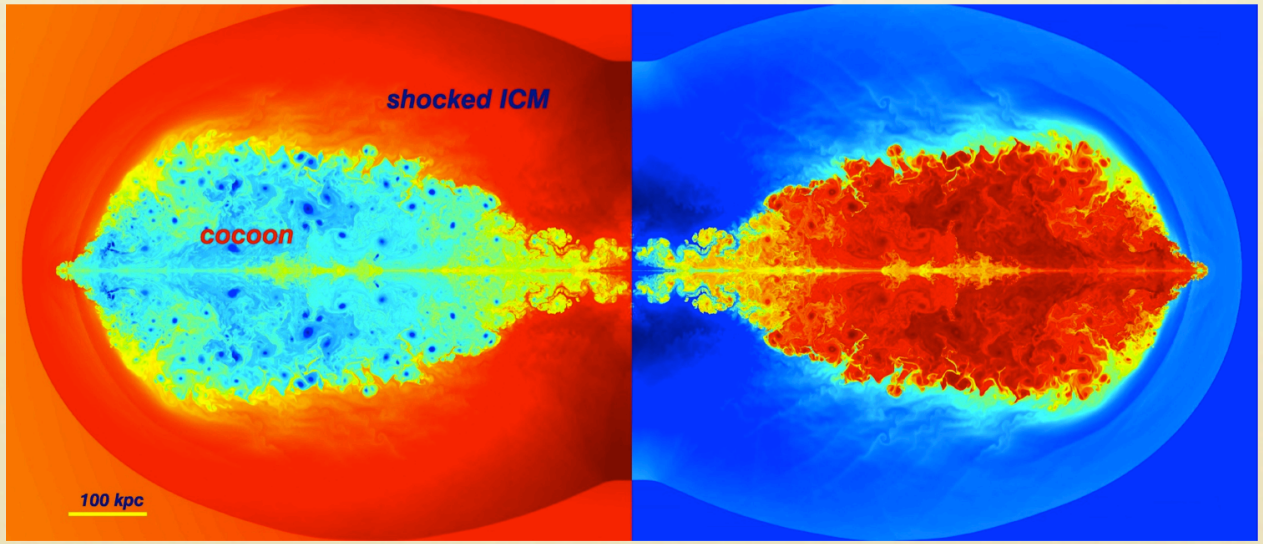
- ★ Approximate Riemann Solvers
- ★ Inter-cell reconstruction techniques
- ★ A method of lines based on a Total Variation Diminishing technique to provide up to third order accuracy both in time and in space.



The code can also be used with the SPEV code, which can account for the evolution of non-thermal particles using finite difference methods which are implicit or semi-implicit. (See talk of Petar Mimica for details of SPEV)

RATPENAT

- Relativistic jets are a common feature of radio-loud Active Galactic Nuclei and X-ray Binary Star Systems. We investigate the physics of extragalactic relativistic jets in several different contexts.
- We focus on the evolution of relativistic flows and their interaction with their environments.
- For this, we need three-dimensional simulations with the largest possible resolution. Thus, we need to test the upgraded version of Ratpenat, which includes 3D parallelization.
- We will perform different tests of scalability of the new code and will try to check the optimal resolution needed to study this problem.



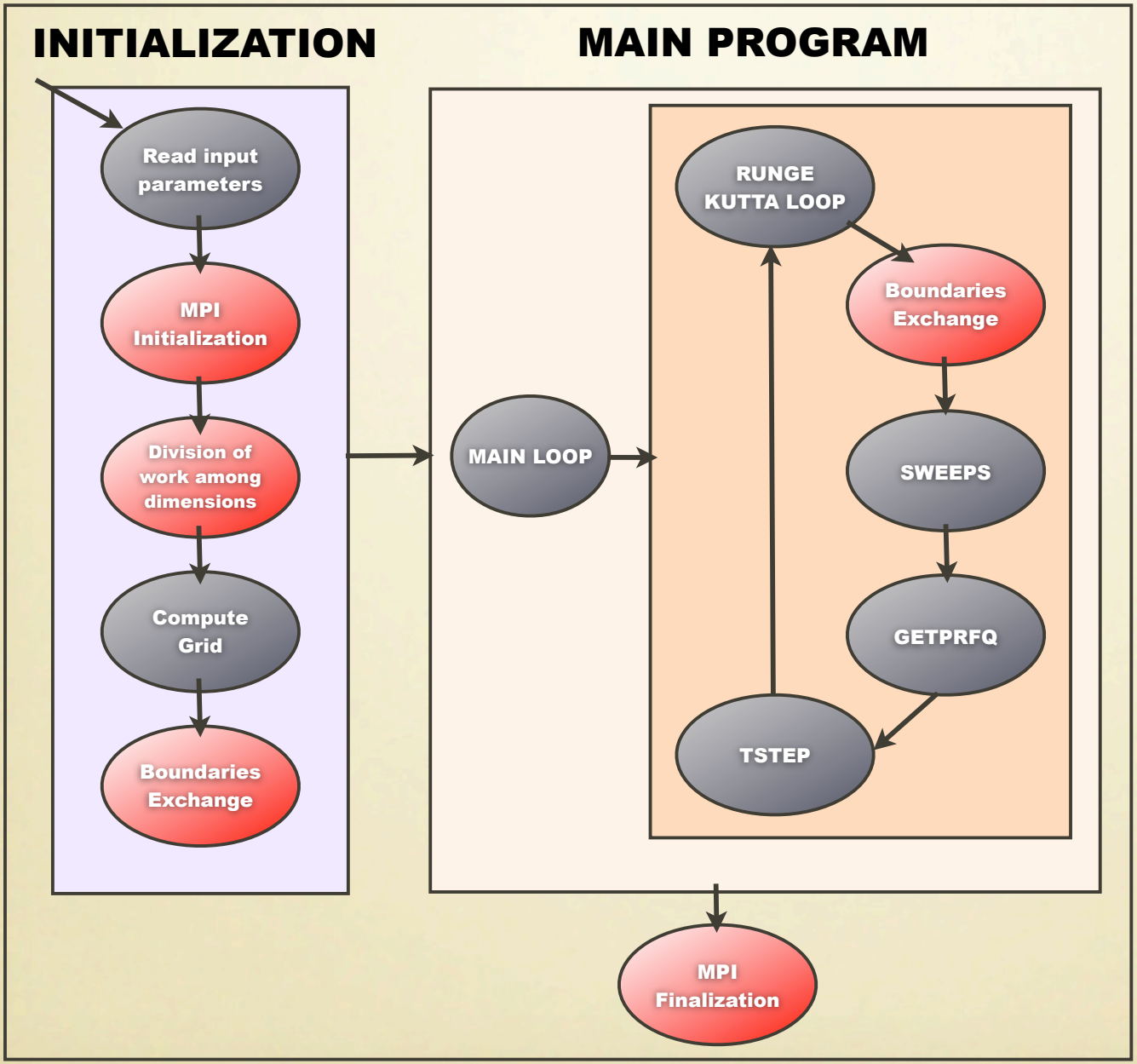
IMPLEMENTATION

- **MRGENESIS and RATPENAT are implemented in Fortran 90**
- **Parallel I/O is obtained by using the Hierarchical Data Format (HDF5).**
- **Parallelized with a hybrid MPI / OpenMP model.**

MPI is a Message Passing Interface which is used widely on parallel machines with *distributed memory*.

OpenMP is an Application Program Interface (API), is used on parallel machines with *shared memory*.

DATA FLOW DIAGRAM



The red nodes are modules of the application exclusively added to parallelize the code

Parallelization

MPI initialization

```
!   MPI initialization
    call MPI_INIT( ierr )

!   create a Cartesian MPI world
    call MPI_CART_CREATE(MPI_COMM_WORLD, maxDims, &
    mpi_dims(1:maxDims), periods(1:maxDims), .true.,  &
    CART_WORLD, ierr)

!   get Cartesian rank
    call MPI_COMM_RANK(CART_WORLD, cartrank, ierr)

!   get Cartesian coordinates
    call MPI_CART_COORDS(CART_WORLD, cartrank, maxDims, &
    mpi_coords, ierr)

!   get the rank of the neighbors
    call MPI_CART_SHIFT( CART_WORLD, 0, 1, xleft, xright, ierr )
    call MPI_CART_SHIFT( CART_WORLD, 1, 1, yleft, yright, ierr )
    call MPI_CART_SHIFT( CART_WORLD, 2, 1, zleft, zright, ierr )
```

Domain decomposition

```
nx = nx / xCPUs
ny = ny / yCPUs
nz = nz / zCPUs
```

Parallelization

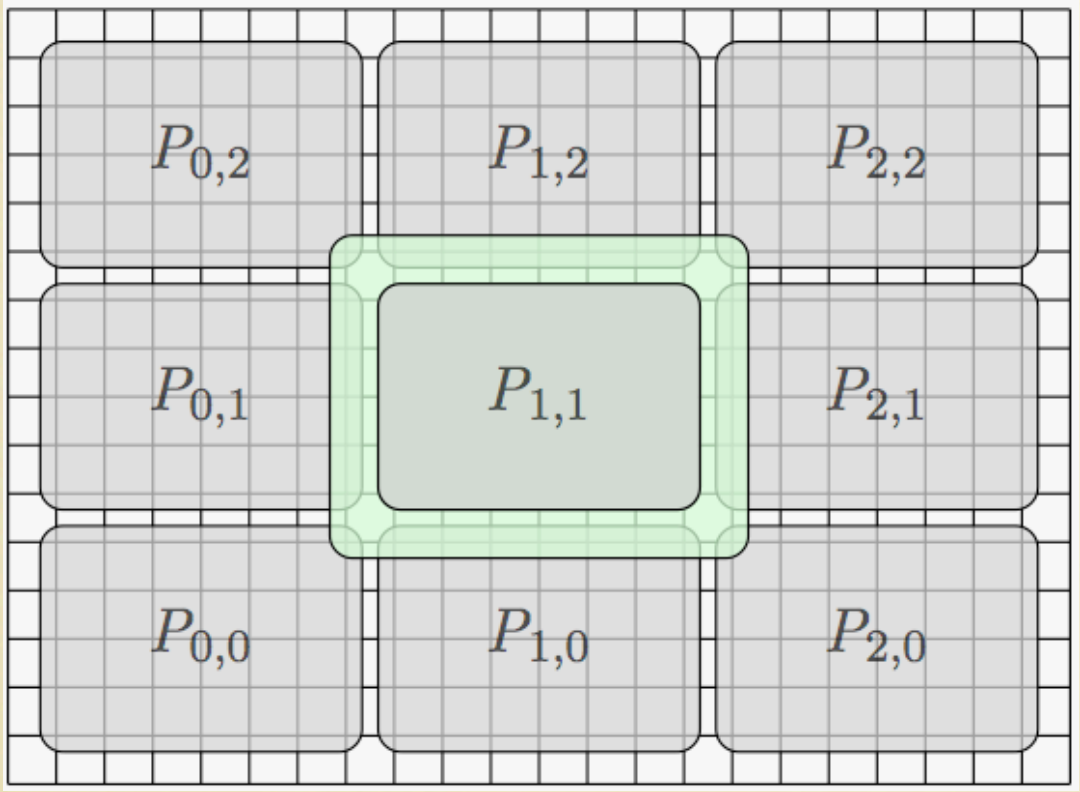
Division of work among dimensions

```
subroutine reparto_trabajo( longitud(1), longitud(2), longitud(3), ...)

  DO i=1,3
         bsize = longitud(i) / mpi_dims(i)

         if (bsize < 5) then
            call ERROR( ... )
         endif

         iniTramo(i,0)  = 1
         finTramo(i,0)  = bsize
         sizeTramo(i,0) = bsize

         do pp= 1, mpi_dims(i)-1
            iniTramo(i,pp)  = finTramo(i,pp-1) + 1
            finTramo(i,pp)  = finTramo(i,pp-1) + bsize
            sizeTramo(i,pp) = bsize

          enddo

         sizeTotal(i) = longitud(i)
         longitud(i)  = sizeTramo(i,mpi_coords(i))

    ENDDO
(...)
END subroutine reparto_trabajo
```

- ★ **longitud**: vector of 3 elements with the size of the grid by dimension
- ★ **mpi_dims**: vector of 3 elements with the number of threads by dimension
- ★ **iniTramo**: matrix with the index of the first element of the grid by thread by dimension
- ★ **finTramo**: matrix with the index of the last element of the grid by thread by dimension
- ★ **sizeTramo**: matrix with the size of the grid by thread by dimension

Parallelization

Boundaries exchange



- ★**P**: MPI thread
- ★**index i, j**: dimension X, Y in cartesian coordinates of the communicator.
- ★**Each Pi,j computes the operations for each element inside of the square.**
- ★**To compute de boundaries of the squares, Pi,j needs to know the value of the neighbors (5 cells):**
  - ★**i-1**
  - ★**i+1**
  - ★**j-1**
  - ★**j+1**
- ★**Solution: Send to neighbors the 5 first cells and receive from them this "halo", store them into the "gosth cells" and then compute the operations.**

Parallelization

Boundaries exchange

```
! Initialize
    lon = 5*4
    req_cnt = 0

! Exchange right side
! Exchange xzn/dx right side
      if (mpi_coords(1).ne.mpi_dims(1) - 1) then
         do ii= -4, 0
            boundsXaux(ii)%xzn  = xzn(nx+ii)
            boundsXaux(ii)%xznl = xznl(nx+ii)
            boundsXaux(ii)%xznr = xznr(nx+ii)
            boundsXaux(ii)%dx   = xznl(nx+ii+1) - xznl(nx+ii)
         enddo
         req_cnt = req_cnt + 1
         call MPI_ISEND( boundsXaux(-4), lon, MPI_DOUBLE_PRECISION, xright, &
         tagxright, CART_WORLD, arr_request(req_cnt), ierr )
         req_cnt = req_cnt + 1
         call MPI_IRECV( boundsX(1), lon, MPI_DOUBLE_PRECISION, xright, &
         tagxleft, CART_WORLD, arr_request(req_cnt), ierr )
      endif

 ! Exchange yzn/dx right side
      if (mpi_coords(2).ne.mpi_dims(2) - 1) then
        (...)
      endif

! Exchange zzn/dx right side
      if (mpi_coords(3).ne.mpi_dims(3) - 1) then
        (...)
      endif
```

Parallelization

Boundaries exchange

```
! Exchange left side
! Exchange xzn/dx left side
   if (mpi_coords(1).ne.0) then
      do ii= 1, 5
         boundsXaux(ii)%xzn  = xzn(ii)
         boundsXaux(ii)%xznl = xznl(ii)
         boundsXaux(ii)%xznr = xznr(ii)
         boundsXaux(ii)%dx   = xznl(ii+1) - xznl(ii)
      enddo
      req_cnt = req_cnt + 1
      call MPI_ISEND( boundsXaux(1), lon, MPI_DOUBLE_PRECISION, xleft, tagxleft, CART_WORLD, &
      arr_request(req_cnt), ierr )
      req_cnt = req_cnt + 1
      call MPI_IRECV( boundsX(-4), lon, MPI_DOUBLE_PRECISION, xleft, tagxright, CART_WORLD, &
      arr_request(req_cnt), ierr )
   endif

! Exchange yzn/dx left side
   if (mpi_coords(2).ne.0) then
        (...)
   endif

! Exchange zzn/dx right side
   if (mpi_coords(3).ne.0) then
       (...)
   endif

! wait for all communication to finish
  call MPI_WAITALL(req_cnt, arr_request, arr_status, ierr)
```

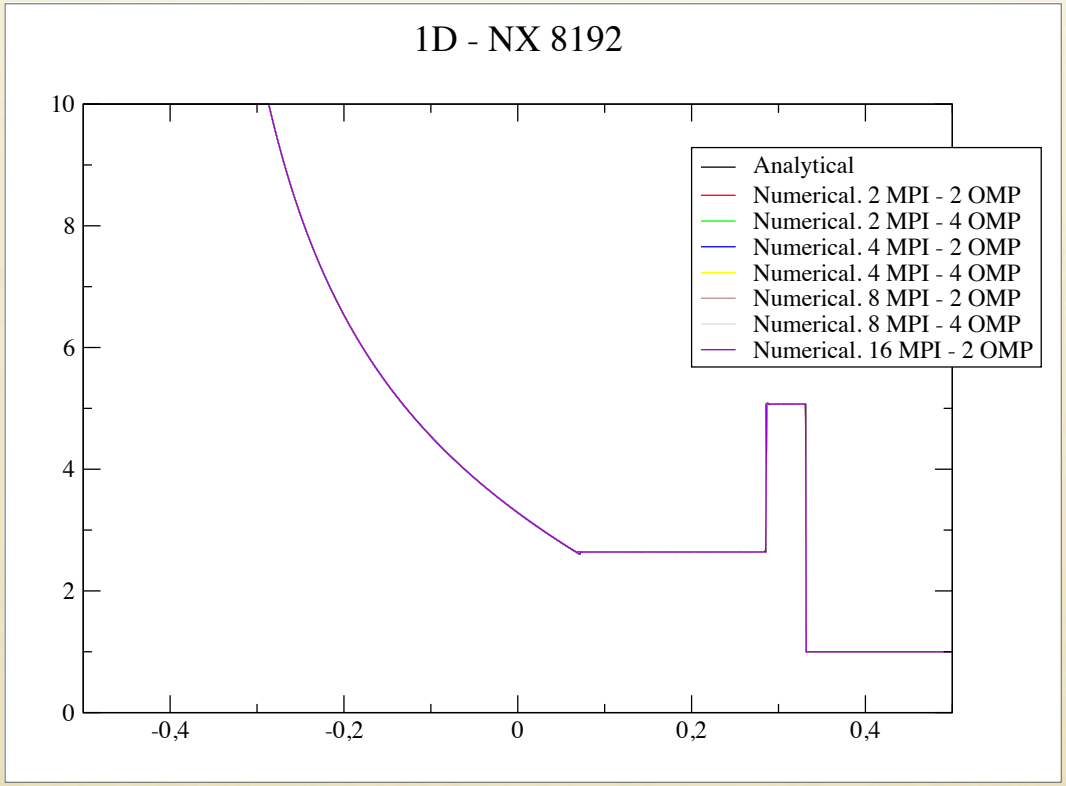
How to compare sequential and parallel codes?

- **Perform validation tests**: Check that ***the results are the same up to the truncation error*** of the algorithm.
  - it is possible that the parallel and sequential codes have different algorithms, which should converge to the same order of accuracy.
  - roundoff errors accumulate in the numerical solution differently in parallel and sequential codes (because of, e.g., the difference in the order of the operations).
- **Perform scalability tests**: how to perform them?, what to compare?
  - Weak scaling
  - Strong scaling
  - Mixture of both (our approach).

MRGENESIS VALIDATION TESTS

Tests to check that the parallel code obtains *the same* results as sequential code:

1) Local comparison: confront the **analytical** and the **numerical** solution of selected test beds.



MRGENESIS VALIDATION TESTS

Tests to check that the parallel code obtains *the same* results as sequential code:

2) Global comparison: Calculating the integral and getting the **relative error** between analytical and numerical solutions.

HYBRID CODE MPI-OMP: CALCULATE RELATIVE ERROR BETWEEN ANALYITIC AND NUMERICAL SOLUTION FOR NX = 1000

Analytic integral: 4.937786859999255

Numerical integral (2 MPI x 4 OpenMP): 4.937660410999951
RELATIVE ERROR: **2.370E-002** 😊

Numerical integral (4 MPI x 2 OpenMP): 4.937658423999932
RELATIVE ERROR: **2.376E-002** 😊

Variations among different partitions (OpenMP-MPI) originate from the differences in the roundoff errors, which account for the differences in the relative errors. 😟

The results are the same up to truncation error! 😊

MRGENESIS VALIDATION TESTS

Tests to check that the parallel code obtains *the same* results as sequential code:

3) Computing the **order of the method** from the numerical results with progressively finer grids.

HYBRID CODE MPI-OMP: CALCULATE THE ORDER OF THE METHOD

EXPECTED ORDER 1 (due to the presence of shocks in the solution)

Relative error (2 MPI x 4 OpenMP, 1000 points): 2.370E-002
Relative error (2 MPI x 4 OpenMP, 2000 points): 1.247E-002
ORDER: **0.9269168246445296** 😊

Relative error (4 MPI x 2 OpenMP, 1000 points): 2.376E-002
Relative error (4 MPI x 2 OpenMP, 2000 points): 1.250E-002
ORDER: **0.9263252410623309 4** 😊

SCALABILITY TESTS ON MARE NOSTRUM (BSC)

- MareNostrum has 31 racks with 6 Blade Centers per rack.
- A Blade Center has 14 Server Blade JS21, which has 2 dual-core processors PowerPC 970MP@2.3 GHz with 1 MB of L2 cache.
- Blade nodes JS21 are interconnected through Myrinet.

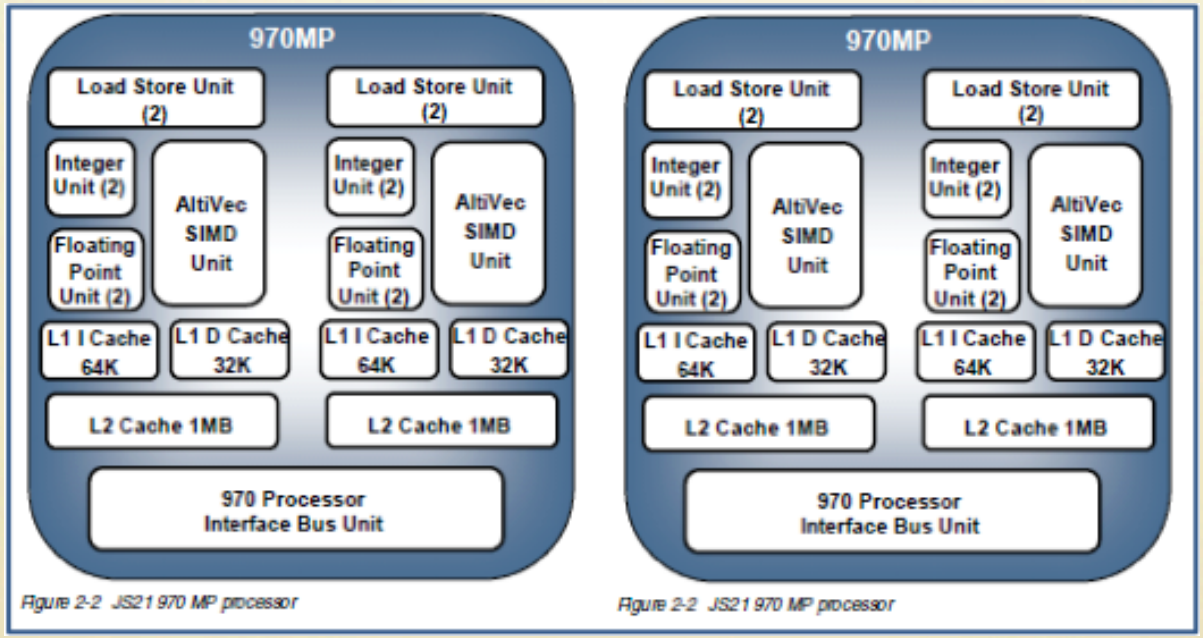


Figure 2-2 JS21 970 MP processor

Figure 2-2 JS21 970 MP processor

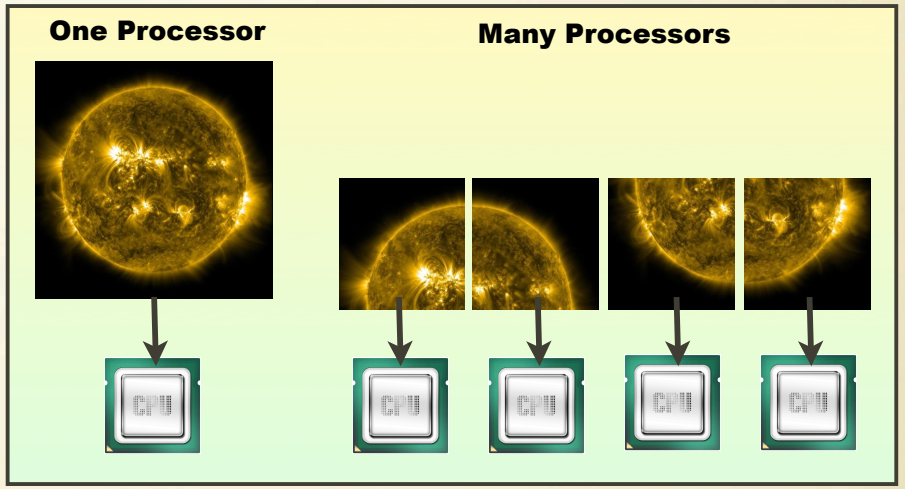
see talk of David Vicente

SCALABILITY TESTS

STRONG SCALING TESTS:

Run a **fixed-size** problem on a **varying number of processors** to see how the timing of the computation scales with the number of processors.

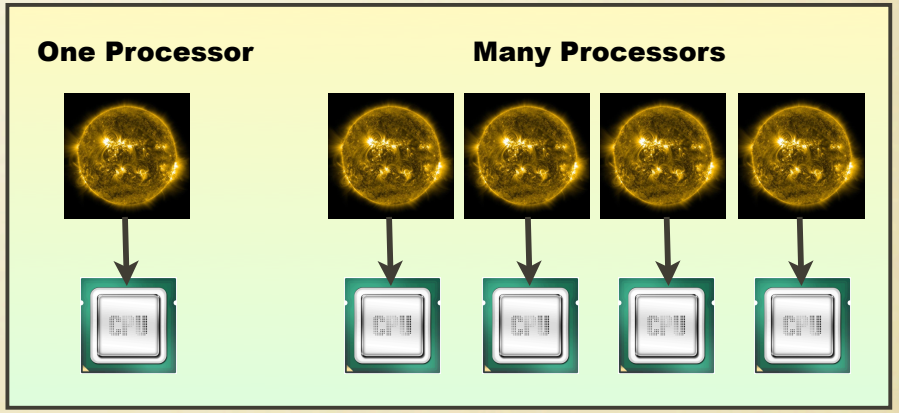
Tells us *how the parallel overhead behaves as the number of cores grows*.



WEAK SCALING TESTS:

Fixes the amount of **work per processor** and compares the execution time over number of processors. Since each processor has the same amount to do, in the ideal case the execution time should remain constant.

Tells us *whether the parallel overhead varies faster or slower than the amount of work*.



SCALABILITY TESTS ON MARE NOSTRUM (BSC)

STRONG SCALING TESTS:

We ran a **standard hydrodynamic problem** (the interaction of stellar winds) with different numbers of processors and combinations of MPI and OpenMP threads. This is a real case of use frequently executed in production environments.

TEST SET UP (MEDIUM SIZED PROBLEM)

GRID: 2880 x 2880
Compiler: IBM XLF90
Optimization options: -O3 -qstrict -q64 -qtune=ppc970 -qarch=ppc970 -qcache=auto
Iterations: 200
SpeedUp relative to 16 CPUs.

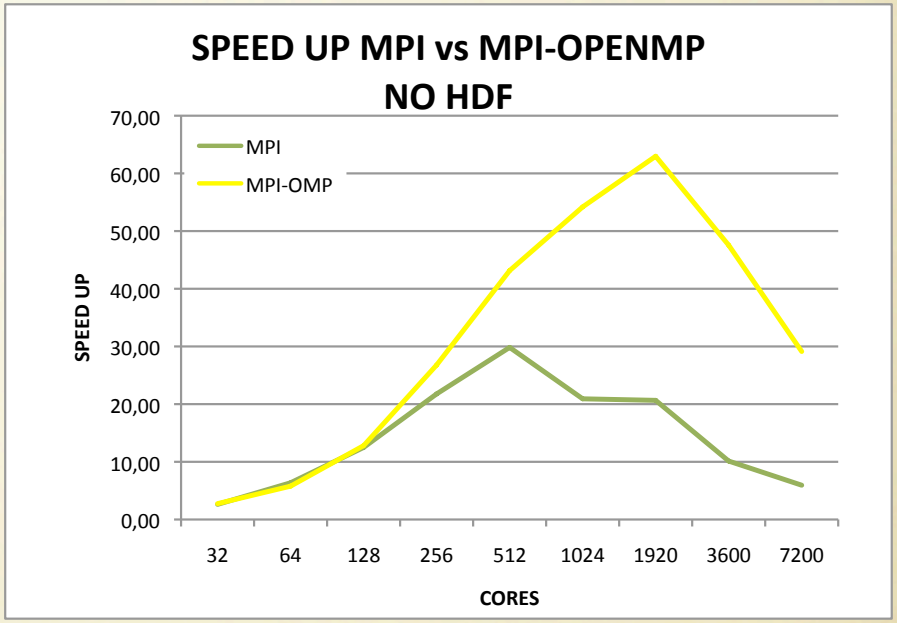
TEST SET UP (LARGE PROBLEM)

GRID: 8640 x 8640
Compiler: IBM XLF90
Optimization options: -O3 -qstrict -q64 -qtune=ppc970 -qarch=ppc970 -qcache=auto
Iterations: 600
SpeedUp relative to 64 CPUs.

SCALABILITY TESTS ON MARE NOSTRUM (BSC)

RESULTS OF MEDIUM SIZED PROBLEM

- The pure MPI model runs faster because the latency introduced by the management of OpenMP threads is greater than the latency for MPI on a small number of processors.
- However, *when the number of processors increases*, the message passing latency grows and penalizes significantly the execution time compared to the hybrid model.
- The hybrid model saturates with more processors, 1920 processors, while the pure MPI model saturates with 512.



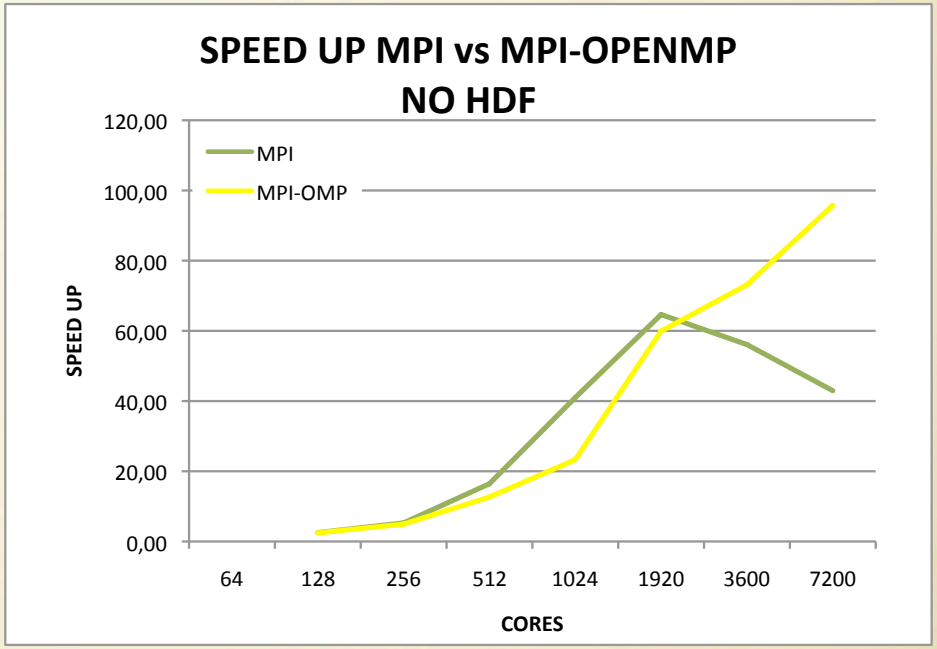
2880x2880	WIND	maxiter: 200		
CPUS	MPI	OPENMP	TIME	SPEEDUP
16	16	1	1.611,00	
32	32	1	614,00	2,62
64	64	1	253,00	6,37
128	128	1	129,00	12,49
256	256	1	74,00	21,77
512	512	1	54,00	29,83
1024	1024	1	77,00	20,92
1920	1920	1	78,00	20,65
3600	3600	1	159,00	10,13
7200	7200	1	271,00	5,94

2880x2880	WIND	maxiter: 200		
CPUS	MPI	OPENMP	TIME	SPEEDUP
16	4	4	2.330,00	
32	8	4	847,00	2,75
64	16	4	403,00	5,78
128	32	4	182,00	12,80
256	64	4	87,00	26,78
512	128	4	54,00	43,15
1024	256	4	43,00	54,19
1920	480	4	37,00	62,97
3600	900	4	49,00	47,55
7200	1800	4	80,00	29,13

SCALABILITY TESTS ON MARE NOSTRUM (BSC)

RESULTS OF LARGE PROBLEM

- The mesh size has a significant impact on scalability.
- A larger mesh size increases the work-per-core, reduces the ratio number of numerical zones to be communicated across domains to number of zones in each domain.
- The **hybrid code keeps scaling (though sub-linearly) up to 7200 processors**.



8640x8640	WIND	maxiter: 600		
CPUS	MPI	OPENMP	TIME	SPEEDUP
64	64	1	19.071,00	
128	128	1	7.563,00	2,52
256	256	1	3.608,00	5,29
512	512	1	1.159,00	16,45
1024	1024	1	465,00	41,01
1920	1920	1	295,00	64,65
3600	3600	1	340,00	56,09
7200	7200	1	444,00	42,95

8640x8640	WIND	maxiter: 600		
CPUS	MPI	OPENMP	TIME	SPEEDUP
64	16	4	18.577,00	
128	32	4	7.563,00	2,46
256	64	4	3.752,00	4,95
512	128	4	1.468,00	12,65
1024	256	4	797,00	23,31
1920	480	4	310,00	59,93
3600	900	4	254,00	73,14
7200	1800	4	194,00	95,76

ANALYSIS WITH THE HPC TOOL PARAVER

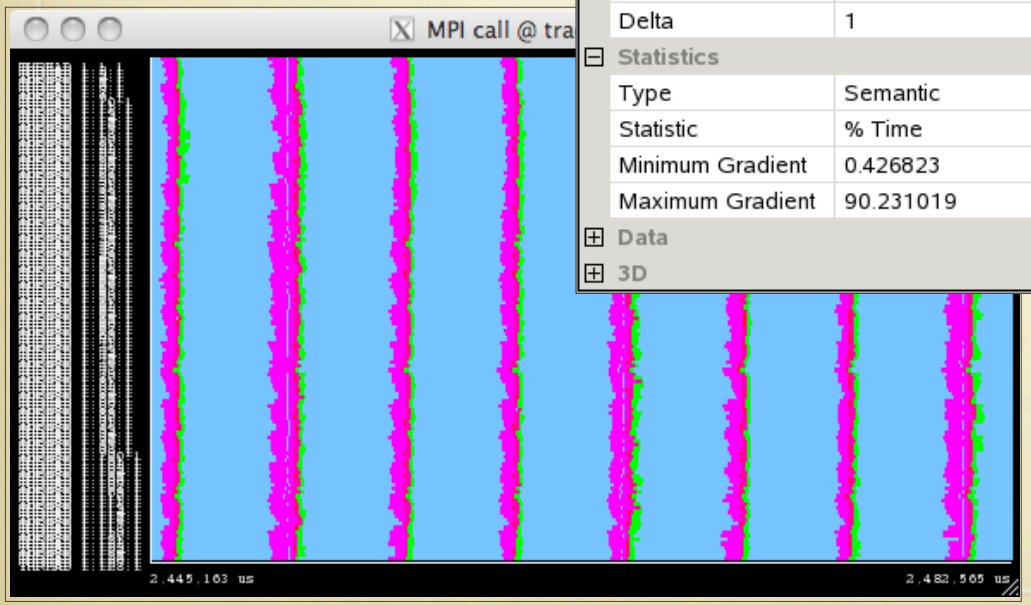
- **Paraver** is part of the CEPBA-Tools toolkit
- **Goal**: a qualitative global perception of the application behavior by visual inspection and then to be able to focus on the detailed quantitative analysis of the problems
- **Steps** to configure the traces that Paraver needs to analyze the execution:
  - Compile the code linking with trace libraries: lmpitracef, lpapi
  - Write *mpitrace.xml*
  - Make a shellscript *trace.sh* several exports of environment variables and call it from the script used to launch the jobs on the Mare Nostrum queues.
  - Join the traces
  - Run **Paraver**!! 😊

```
<counters enabled="yes">
  <cpu enabled="yes">
    <set enabled="yes" domain="all">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L1_DCM
    </set>
    <set enabled="yes" domain="all">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L3_DCM
    </set>
  </cpu>
</counters>
```

ANALYSIS WITH THE HPC TOOL PARAVER

Analysis of communications with 128 MPI

% Time in MPI Calls
vs
% time doing operations



Window properties

Name	New Histogram #2
Begin time	2,445,163.05 us
End time	2,482,565.68 us
Control	
Window	MPI call
Minimum	0
Maximum	32
Delta	1
Statistics	
Type	Semantic
Statistic	% Time
Minimum Gradient	0.426823
Maximum Gradient	90.231019
Data	
3D	

New Histogram #2 @ trace.prv

	End	MPI_Isend	MPI_Irecv	MPI_Waitall	MPI_Allreduce
THREAD 1.107.1	82.81 %	2.37 %	1.16 %	1.53 %	12.13 %
THREAD 1.108.1	81.61 %	2.36 %	1.13 %	3.47 %	11.44 %
THREAD 1.109.1	85.17 %	2.32 %	1.16 %	2.55 %	8.80 %
THREAD 1.110.1	85.26 %	2.13 %	1.26 %	1.41 %	9.95 %
THREAD 1.111.1	86.03 %	2.19 %	1.16 %	1.40 %	9.21 %
THREAD 1.112.1	83.30 %	1.60 %	0.94 %	3.18 %	10.98 %
THREAD 1.113.1	80.17 %	1.67 %	1.04 %	2.18 %	14.94 %
THREAD 1.114.1	82.37 %	2.28 %	1.21 %	1.98 %	12.15 %
THREAD 1.115.1	81.92 %	2.23 %	1.11 %	1.18 %	13.57 %
THREAD 1.116.1	81.79 %	2.42 %	1.08 %	3.04 %	11.67 %
THREAD 1.117.1	84.86 %	2.33 %	1.05 %	3.23 %	8.53 %
THREAD 1.118.1	88.14 %	2.21 %	1.17 %	1.84 %	6.63 %
THREAD 1.119.1	90.23 %	2.28 %	1.03 %	2.66 %	3.80 %
THREAD 1.120.1	84.73 %	1.61 %	0.93 %	3.72 %	9.01 %
THREAD 1.121.1	79.38 %	1.01 %	0.47 %	4.06 %	15.07 %
THREAD 1.122.1	80.92 %	1.30 %	0.87 %	2.42 %	14.49 %
THREAD 1.123.1	82.78 %	1.17 %	0.96 %	3.91 %	11.18 %
THREAD 1.124.1	80.43 %	1.29 %	0.74 %	3.58 %	13.97 %
THREAD 1.125.1	82.71 %	1.40 %	0.77 %	4.00 %	11.12 %
THREAD 1.126.1	84.23 %	1.23 %	0.89 %	3.93 %	9.72 %
THREAD 1.127.1	84.02 %	1.40 %	0.79 %	3.59 %	10.20 %
THREAD 1.128.1	82.96 %	0.99 %	0.43 %	4.14 %	11.48 %
Total	10,678.55 %	255.63 %	136.71 %	360.56 %	1,368.55 %
Average	83.43 %	1.00 %	1.07 %	2.82 %	10.69 %
Maximum	90.23 %	2.45 %	1.46 %	5.98 %	15.27 %
Minimum	79.38 %	0.98 %	0.43 %	0.80 %	3.80 %
StDev	2.16 %	0.39 %	0.16 %	1.10 %	2.35 %
Avg/Max	0.92	0.82	0.73	0.47	0.70

92% of Avg/max: 8% of time in communications

ANALYSIS WITH THE HPC TOOL PARAVER

Analysis of communications: IPC comparison between 128 MPI and 32 MPI threads

Window properties	
Name	New Histogram #2
Begin time	2,379,172.26 us
End time	2,402,289.62 us
Control	
Window	Instructions per cycle
Minimum	0.037283
Maximum	1.131928
Delta	2
Statistics	
Type	Semantic
Statistic	Average value
Minimum Gradient	0.951574
Maximum Gradient	1.037229
Data	
3D	

New Histogram #2 @ trace.prv

	[0.0372828..1.13193]
THREAD 1.19.1	1.01
THREAD 1.20.1	1.01
THREAD 1.21.1	1.01
THREAD 1.22.1	1.02
THREAD 1.23.1	1.01
THREAD 1.24.1	1.02
THREAD 1.25.1	1.03
THREAD 1.26.1	1.01
THREAD 1.27.1	1.02
THREAD 1.28.1	1.01
THREAD 1.29.1	1.02
THREAD 1.30.1	1.01
THREAD 1.31.1	1.00
THREAD 1.32.1	0.98
Total	32.31
Average	1.01
Maximum	1.04
Minimum	0.95
StDev	0.02
Avg/Max	0.97

New Histogram #1 @ trace.p

	[0.0591388..1.12077]
THREAD 1.116.1	1.02
THREAD 1.117.1	1.00
THREAD 1.118.1	0.98
THREAD 1.119.1	0.97
THREAD 1.120.1	1.01
THREAD 1.121.1	1.03
THREAD 1.122.1	1.03
THREAD 1.123.1	1.02
THREAD 1.124.1	1.02
THREAD 1.125.1	1.02
THREAD 1.126.1	1.01
THREAD 1.127.1	1.01
THREAD 1.128.1	1.03
Total	129.16
Average	1.01
Maximum	1.03
Minimum	0.97
StDev	0.01
Avg/Max	0.98

The code efficiency is maintained even increasing the number of threads.

Average of IPC is 1.01 in both ecutions.

CONCLUSIONS

- ★ Due to complexity of relativistic hydrodynamics codes and of their parallel implementation, the collaboration between astrophysicists and computer scientists is needed in order to get the best efficiency and be able to run realistic problems on the fastest supercomputers.
- ★ The results obtained with the hybrid OpenMP/MPI parallelization of MRGENESIS, indicate a significant improvement of **efficiency** that allows us run the code optimally **up to 7200 processors**.
- ★ ON GOING: We will perform different tests of scalability of the new RATPENAT code and will try to check the optimal resolution needed to study 3D problems.

THANKS FOR YOUR ATTENTION!!